Automated System Testing for a Learning Management System

https://doi.org/10.3991/ijet.v15i24.12073

Lukas Krisper, Markus Ebner, Martin Ebner (✉)
Graz University of Technology, Graz, Austria
martin.ebner@tugraz.at

Abstract—Over the last years software development life-cycles have continuously been shortened and new releases are being deployed at a more and more frequent level. In order to ensure the quality of those releases, a strong shift towards automated testing at all testing levels has become noticeable throughout the software development industry. At system testing level, the scope of testing is the developed product as a whole, tested in a test environment that has a very close resemblance to the production system. Because of this system-wide scope and the many potential sources for failures, the implementation of automated tests at this level is challenging. Exhaustive testing is neither feasible nor maintainable, therefore proper designed test cases that cover important functionality are essential. Due to increasing laws and regulations on data protection and data privacy, proper management of test data used in automated testing is as important. This paper discusses how automated system tests for TeachCenter 3.0, Graz University of Technology's learning management system, were implemented.

Keywords—Automation, system testing, regression, learning management system, test data, test cases

1 Introduction

Graz University of Technology rolled out a new release of the university's Learning Management System (LMS) called "TeachCenter 3.0" in August 2019. In order to ensure that essential use cases can still be performed by teachers and students in the new release, automated tests where implemented. According to the International Software Testing Qualifications Board (ISTQB), test automation is defined as using software to either support or perform testing activities. In test automation, software is used not only for test execution per se, but also for activities like management of test cases, design of test cases or the evaluation and reporting of test results [1]. While initially seen as a possibility to increase efficiency and reduce costs, test automation has become an essential part of software development processes over the last years [2]. This paper focuses on the following research questions (RQx):

- (RQ1) How can the existing core functionality of a large software system be assured automatically in new versions of the system?
- (RQ2) Which test cases and test suites need to be designed to test a learning management system effectively?
- (RQ3) Which test data is needed to test a learning management system and how can the data be provided to automated test cases?

1.1 Testing level

Software development usually is conducted after a predefined development model like waterfall model, spiral model or various agile models. Each of those models contains an idea how tests should be performed, but testing according to the principles of the so called general V-model can be applied to the other models. Therefore, the general V-model holds a special position within the models. The general V-model is illustrated in Fig 1and differentiates between following testing levels: Component testing, integration testing, system testing, and acceptance testing. Each testing level puts a different focus on the System Under Test (SUT). The test cases discussed in this paper were implemented at system testing level. At this level, the developed software product is considered in its entirety and tested in an environment that has a close resemblance to the production environment. Tests are conducted from the customer's or user's perspective and validate if the software has been implemented according to the requirements. Tests on system testing level are typically performed using the systems General User Interface (GUI) to interact with the system from a user's perspective [3]. Tests at system testing level should confirm the functioning of the GUI [4], ensure that the system meets business requirements, is stable and in a state for manual testing to be reasonable [5]. Furthermore, tests at system testing level can be seen as a second line of defence, as failures in higher testing levels additionally show that tests at lower testing levels like integration testing or component testing are incorrect or missing [6].

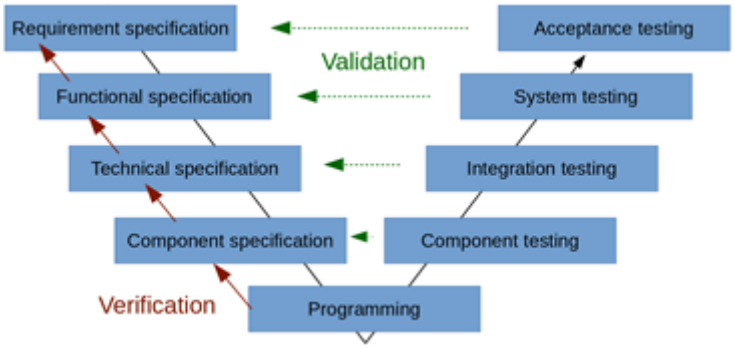

Fig. 1. General V-model (based on [3], p.42)

1.2 Testing type

Spillner and Linz [3] differentiate between four basic types of testing: Functional testing, non-functional testing, structural testing and testing related to changes.

Regression tests are categorised as a type of tests which are performed in order to ensure that already existing functionality of a software is still present after changes were made to the software [3]. Independent of the testing level, regression tests are suitable to be one of the first types of tests to be automated in a software project. One of the main advantages of automated testing compared to manual testing is the possibility to run a set of tests to ensure that the code changes made did not break any functionalities of the software [7].

Furthermore, the execution of automated regression tests is more reliable compared to a manual execution of regression tests [8]. As TeachCenter 3.0 is a new version of an already existing software system, regression tests were the focus of the test automation activities in order to ensure that changes to the software do not have a negative effect on the already present functionality and that essential use cases can still be performed after the application of the changes.

2 Test Cases for a LMS

An important aspect of test automation is the selection of the test cases to be automated. If this selection is not done properly, the automation of test cases would result in being able to quickly execute test cases, which have no value, on a regular basis. The following steps were performed in order to identify the test cases to automate for TeachCenter 3.0: A review of the literature of different LMS [9][10][11] was performed in order to identify important use cases. An interview with the first level support of TeachCenter 2.0 was conducted in order to find out whether important use cases at Graz University of Technology were similar to those of other educational institutions. The LMS that is used at Graz University of Technology supports the generation of statistics on which features of the LMS are used to what extent. Those statistics corresponded to the results of the interview. Despite the different LMS used by different universities, the use cases that are being considered important are similar. Those use cases contain activities like reading the course's contents, communicating with other participants or submitting material to the course. In total about 30 test cases were created based on those use cases and combined to test suites.

The test cases are written in a "Given-When-Then"-structure which is commonly used in Behaviour-Driven-Development (BDD) in order to strengthen the focus on the user and the system's behaviour: Given a certain precondition, when a certain action is performed, then the following results are expected. An example can be found in Table 1.

Table 1. Example of Given-When-Then Structure

Given	**User is on a blank browser page**
When	User opens index.php of TeachCenter 3.0
Then	TeachCenter front page is shown and fully loaded

This style is used in the creation of test cases without pursuing BDD in order to avoid a level of unnecessary complexity and overhead when implementing regression tests [12]. Frameworks for BDD serve the purpose of facilitating communication between people of different backgrounds (like customers, project managers, business administrators, developers or testers) when discussing new features in a software project. The resulting definitions of new features are used for validation once the new features were implemented. The type of tests discussed in this paper are regression tests – tests that ensure that given functionality is still present after changes have been made to the software. The features that are covered by those tests are already set and there is no need to use BDD in order to specify them with various stakeholders. This is also one of the reasons why Behat tests, which are part of Moodle LMS, were not used in the project.

3 Test Data for a LMS

Albrecht-Zölch [13] differentiates between two basic types of data that can be used for testing: real data and synthetic data. Real data is data which is taken from a production system and transferred to a test system. Synthetic data is data which is solely created for the purpose of testing.

Data protection regulations like the General Data Protection Regulation (GDPR) [14] have a huge impact on the usage of test data in software testing as they restrict the usage of real data in testing. All personal data that is present in the set of real data has to be anonymised before using real data in testing. The SUT in this paper is a LMS. A LMS basically contains data on courses, data on people involved in the courses as well as data that result from the interaction of people with the courses. In case of a LMS which typically contains teachers as well as student's names, email addresses or identification numbers, personal data that has to be anonymised according to the GDPR. Data like course descriptions or learning materials usually does not contain personal data, although personal data is not always easy to detect and therefore not always easy to anonymise (e.g. if a student included personal data in an assignment that was submitted to the LMS as a PDF-file).

TeachCenter 3.0 is tested with a combination of real data and synthetic data. A test data generator was implemented by the development team using an anonymised set of data retrieved from TUGRAZonline 2.0, the university's campus management system. With this test data generator, test data (e.g. courses or students) can be created on demand. Most of the test data used in the automated system tests was persisted on the SUT and used in the test cases by accessing a component that handles the access on the test data. This way test cases are also separated from test data, which is one of the best practices of using test data in automated testing [13]. Test data that is used exclu-

sively for automated testing is labelled as such in order to not be used in manual testing (e.g. test users for automated tests have names like "Student [TESTAUTOMATION_DO_NOT_TOUCH]" and mail-addresses like "student@ta_d_n_t.ta_d_n_t".)

4 On the Implementation and Infrastructure

Typical phases in test execution stretch from ramp-up (setup, getting ready to run the test) to tear-down (cleanup after tests were run). Those phases are independent from the testing level and testing type and are commonly used across various frameworks in the field [15][16][17]. When structuring tests according to those phases, the tests are designed to be executed in a repeatable way. Besides designing test cases to be executed repeatedly, following other best practices were followed in the implementation of the test cases: Test cases were implemented in order of a breadth-first approach, they are executed in a continuous integration environment and tests have been split into test suits. Tests are kept small and they each test one certain aspect of the LMS. The test cases are stable, independent from each other and were implemented by developing reusable components using design patterns like the Page Object Pattern. Each test has a header that contains further information on the test case itself and each test logs the performed steps for traceability [18][19]. An important aspect of the implementation is the identification of objects in the SUT and the interaction with the SUT.

4.1 Identification of objects

According to Gundecha [20] one of the key success factors when automating tests using the GUI is the identification of the user interface's elements in order to perform actions on those elements as well as verify the results of the actions performed. A stable way of interacting with the GUI is essential. This means being able to execute the tests regardless of screen resolutions, window sizes or language [21]. Elements of the GUI therefore need to be identified properly, using unique (at least in context of the object) and stable features. Features that fulfil that criteria are the ID of an element or the identification of an element via XPath expressions (setting an element's feature in context to other elements of the Document Object Model (DOM)). Table 2contains an example for identification of an element by ID or XPath expression.

Table 2. Identification of elements

DOM Element	**<button type= "button" id= "btn1" data-role= "next "> Next </button >**
ID	btn1
XPath expression	//button[@data-role='next']

4.2 Interaction with SUT

A commonly used design pattern in test automation is the Page Object Pattern. When adopting this pattern, components of web sites are being modelled into reusable objects which offer functionality to interact with the components of the web site [7]. Typical interactions with a web site include actions like clicking elements or entering values to input fields. When implementing a page object, one develops an interface to a web site. By modelling the properties and the behaviour of a web site, the developed interface serves as a layer that separates the actual test code from the code used to interact with the web site [20]. Fig 2 illustrates the creation of page objects from a web site. In this example three page objects are being created: One page object for the header, one page object for the body and one page object for the footer. Each page object bears the functionality to interact with the corresponding part of the site, e.g. the header page object allows the user to navigate to the login page or to switch between languages.

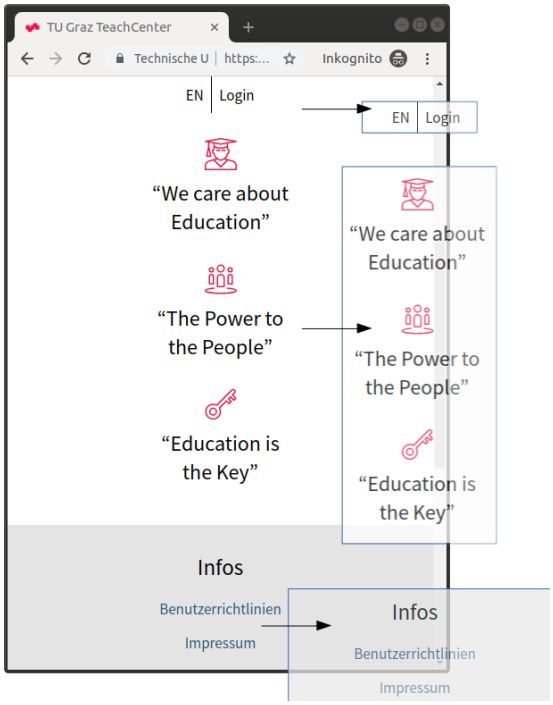

Fig. 2. Creation of page objects

4.3 Implementation

In order to implement page objects according to the Page Object Pattern, the PageFactory, a factory class to initialise the page objects, is being used. By using the PageFactory, all WebElements of a page object are initialised and can be accessed

during testing. WebElements represent elements in the DOM and are part of Selenium WebDriver. Selenium is a suite of tools for automating browsers. Amongst others, Selenium offers functionality to define interactions with elements of a website, e.g. what to click or type. All page objects extend the same superclass which contains functionality that can be used in all page objects like waiting for elements to be present on a page as well as checks if success messages are present. This class also implements the timeouts that are used for waiting for various events during test execution. The individual page objects are used in test classes for interacting with the SUT.

The test classes contain various annotated methods which orchestrate test execution. There do exist methods which are run before the tests (ramp-up) and methods which are run after tests (tear-down) as well as methods for the tests themselves. Altogether the typical phases of test execution are represented in the test classes.

The tests are built and run using Apache Maven. Multiple parameters can be set when running the tests: The test suite, the browser, a runner and the SUT to be tested. The test suite parameter specifies the test cases to be run as well as the level of the logging during test execution (little logging to extensive logging) and the level of parallelisation. The level of parallelisation can be defined by the number of parallel threads that should be used to run tests and by the level (e.g. methods, classes, suits) at which tests should be parallelised. The SUT parameter sets the system to be tested as well as the test data to be used. This allows the test cases to be developed on localhost and run against systems on another host using different sets of test data. Depending on the specified browser parameter, a different instance of WebDriver is created with different browser-specific options. The runner parameter is used to specify how the tests are run depending on the browser. This way different browsers like Firefox or Chrome can be used in different environments (e.g. open a browser window when developing test cases and starting the browser headless when running tests in a continuous integration environment).

4.4 Test infrastructure

Fig 3 contains an overview on the test infrastructure. A client (independent of the device) accesses TeachCenter 3.0 by using a web browser. This access can either happen via Internet or intranet if the device is part of Graz University of Technology's data network (TUGnet). Courses as well as students are synced from a generated test data set based on a clone of the campus management system TUGRAZonline 2.0. A Git repository is used to version control the source code of TeachCenter 3.0 as well as the source code needed for testing. An important part of the test infrastructure is a Jenkins server which pulls the source code of the implemented test cases in certain intervals, builds the tests, runs them against TeachCenter 3.0 and reports the results.

TeachCenter 3.0 is based on version 3.5 of Moodle LMS. A custom theme and plugins developed by Graz University of Technology, as well as additional plugins which are maintained by the Academic Moodle Cooperation are installed on top of the default installation. In addition, several core hacks (modifications of Moodle's source code) were applied. By applying core hacks, a developer changes Moodle's

source code to apply changes that would not be possible by using the ways provided by Moodle itself (e.g. installing plugins or adjusting the configuration). Downsides of core hacks are that they might threaten the stability of the LMS and might not be compatible with future updates.

The test data set must at least contain the following elements for the tests to be run: A course with a section to which resources (PDF-file and a page) and activities (groupchoice, forum, checkmark, scheduler for individual students as well as a scheduler for groups) are added.

Jenkins is used as an automation server. Without the use of an automation server, actions like triggering the automated tests or evaluating the test results must be done manually. Jenkins was chosen as it is a widespread tool that offers many plugins to support a wide variety of frameworks and applications.

As the source code of the tests must be in line with the source code of the SUT (when the SUT changes, tests must be adapted accordingly), also the code of the tests has to be version controlled in order to be able to execute the tests at the needed version. A Git repository was set up in order to achieve this task.

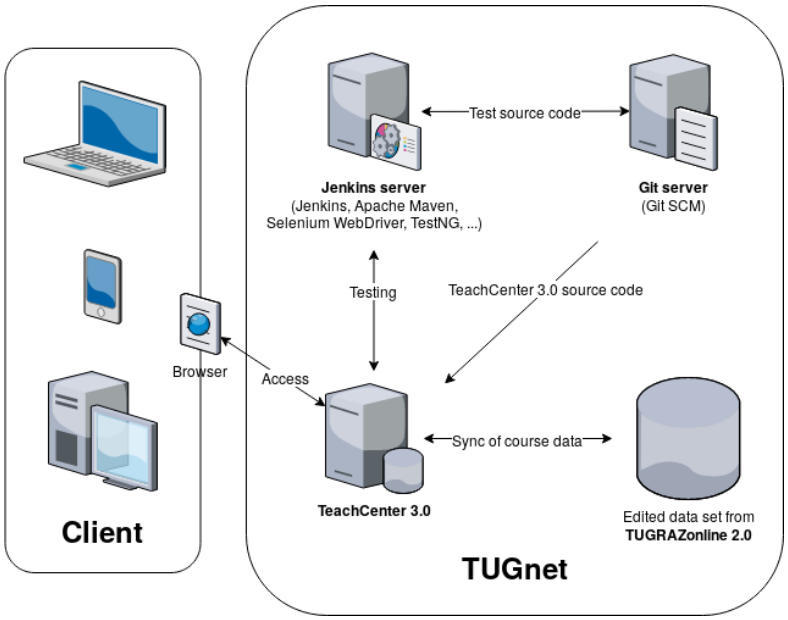

Fig. 3. Overview on test infrastructure

4.5 Operation and maintenance of automated tests

Once implemented, the automated tests must be used in an appropriate way. They must be run regularly, must be integrated into the development process and must be maintained. The implemented tests are run in different execution cycles according to

their test suite. Those execution cycles stretch from test suites that are only run once a week to test suites that are run twice a day.

Before new tests are added to a test suite that is being executed regularly, the tests are added to a test suite that resembles a kind of staging area where the tests are run without having any impact on further builds of the project. In this test suite, the tests must prove their functionality and value in order to become a part of any of the other test suites. This way tests are being refactored and improved until they are stable to advance into another test suite, which should also lower the level of flakiness of the test cases in use. Flaky tests are tests that might fail on one test run and pass on another without any changes to the SUT [22]. The automated test suits of this paper include a listener that listens to the results of individual test cases and if a test case fails it triggers a re-run of the test. Failed tests are re-run up to three times until they are considered as failed.

In order to facilitate the analysis of failed tests, a functionality to take screenshots at the moment a test fails has been implemented. Screenshots are stored with a timestamp, the name of the test method and the name of the browser in a directory that is accessible via the web interface of the Jenkins server. In case of failures in any of the test runs, the failures can be analysed using the test's log output, the stacktrace at the time of the failure as well as the screenshot. The analysis can lead to two possible outcomes: In the first case, the functionality on the SUT is as intended and the result of the test case is incorrect (false negative) which leads to an adaption of the test case. In the second case, the intended functionality on the SUT is not given any more and the test case is correct, which leads to an adaption of the software product.

5 Conclusion and Future Work

After review of the literature and the discussion of different testing levels and types, automated regression tests were implemented on a system testing level according to best practices. A large software system consists of many integrated components that interact with each other. In order to detect possible side effects of changes, a high test level (system testing) was chosen to ensure the stability of core functionalities as tests at this level consider the system as a whole rather than focusing on individual components. A continuous integration environment was installed for regular execution of the implemented tests as well as for reporting of the test results. (RQ1)

In order to identify relevant test cases, literature research was done to find out what essential use cases of systems like learning management systems are. The findings were matched with information provided by TeachCenter's first level support in order to verify the results but also to categorize the use cases according to their relevance. The list of use cases includes activities like providing course materials and submitting assignments. Test cases were implemented based on those use cases and combined to test suites according to their categorisation. (RQ2)

For implementing the test cases only the following test data is needed: A teacher's account, a student's account, a PDF-file and a course with a basic configuration to which both the teacher and the student are enrolled to. The test data in use is mostly

synthetic test data. Some data that exists on the system under test was created by a test data generator that creates test data based on a former set of real data that has been edited according to laws and regulations of data protection and data privacy. Parts of the test data are persisted within the system under test, using identifiers that clearly mark them as test data to be used in automated test cases. Other parts are provided to the test cases by JSON-files that reference the persisted data. This way the implemented test cases are separated from the test data. (RQ3)

ISO 25010 differentiates between eight characteristics that should be considered when evaluating the quality of a software product: Functional Suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability and Portability [23]. This paper only discussed tests for the characteristic of "Functional Suitability", but the established infrastructure could also be used to include tests for other characteristics of software quality. Examples are automated performance tests or automated security tests for a LMS. In general, the level of automation can be increased even if automated tests have already been implemented. One example is the automated creation of page objects. The subject of test data management was mentioned briefly in this paper but contains much more areas to be considered than those that have been covered in this paper. Further research could be done on the application of a whole test data management framework to a LMS. Another step to enhance the automation of tests in general could lie within the omnipresent buzzword "artificial intelligence" or "AI". Bots and machine learning could be used to automatically create basic test cases and could be used to maintain created test cases (e.g. for identifying which tests are not relevant anymore and should be removed from test suites). An advancement in test automation could lead to a focus on improving manual testing activities for complex human behaviour [22].

6 References

[1] ISTQB (2016). ISTQB Glossary. url: https://glossary.istqb.org/en/term/test-automation (visited on 09/07/2019)

[2] Mariani, Leonardo et al. (2017). "The central role of test automation in software quality assurance." In: Software Quality Journal. Springer US, pp. 797–802. https://doi.org/10.1007/s11219-017-9383-5

[3] Spillner, Andreas and Tilo Linz (2015). Basiswissen Softwaretest. Aus- und Weiterbildung zum Certified Tester. Foundation Level nach ISTQB-Standard. third edition. dpunkt.verlag

[4] Cohn, Mike (2009). Succeeding with Agile. Addison-Wesley Professional

[5] Bowles, Ashby (2017). Test Early, Test Often: Automation Testing with the Test Pyramid. url: https://willowtreeapps.com/ideas/test-earlytest-often-automation-testing-with-the-test-pyramid (visited on 09/07/2019) https://doi.org/10.1109/aqtr.2018.8402699

[6] Fowler, Martin (2012). TestPyramid. url: https://martinfowler.com/bliki/TestPyramid.html (visited on 09/07/2019)

[7] Leotta, Maurizio et al. (2013). "Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study." In: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops. IEEE, pp. 108–113. isbn: 978-0-7695-4993-4. https://doi.org/10.1109/icstw.2013.19

[8] Bath, Graham and Judy McKay (2015). Praxiswissen Softwaretest - Test Analyst und Technical Test Analyst. Aus- und Weiterbildung zum Certified Tester - Advanced Level nach ISTQB-Standard. third edition. dpunkt.verlag

[9] Walker, Darrell et al. (2016). "LEARNING MANAGEMENT SYSTEM USAGE, Perspectives From University Instructors." In: Quarterly Review of Distance Education Volume 17 # 2. Information Age Publishing, pp. 41–50

[10] Cantabella, Magdalena et al. (2017). "Searching for Behavior Patterns of Students in Different Training Modalities through Learning Management Systems." In: 2017 International Conference on Intelligent Environments (IE). IEEE, pp. 44–51. https://doi.org/10.1109/ie.2017.31

[11] Dlalisa, Sizwe (2017). "Acceptance and usage of learning management system amongst academics." In: 2017 Conference on Information Communication Technology and Society (ICTAS). IEEE, pp. 1–7. https://doi.org/10.1109/ictas.2017.7920525

[12] Lee, Nick (2017). A BDD Style Regression Test Automation Approach Does Not Make Sense. url: https://medium.com/@nicklee1/a-bdd-style-regression-test-automation-approach-does-not-make-sense-5bc1a682a440 (visited on 09/07/2019)

[13] Albrecht-Zolch, Janet (2018). Testdaten und Testdatenmanagement. Vorgehen, Methoden und Praxis. first edition. dpunkt.verlag

[14] European Commission (2018). Communication from the Commission to the European Parliament and the Council. url: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52018DC0043 (visited on 09/07/2019)

[15] Coyner, Brian M. and Eric M. Burke (2003). Set Up and Tear Down.url: https://www.oreilly.com/library/view/java-extreme-programming/0596003870/ch04s06.html (visited on 09/07/2019)

[16] Reese, John (2018). Unit testing best practices with .NET Core and .NET Standard. url: https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices (visited on 09/07/2019)

[17] Apple Developer Documentation (2019). Understanding Setup and Teardown for Test Methods. url: https://developer.apple.com/documentation/xctest/xctestcase/understanding_setup_and_teardown_for_test_methods (visited on 09/07/2019)

[18] McMahon, Chris (2009). "History of a Large Test Automation Project Using Selenium." In: Proc. 2009 Agile Conference. Chicago, USA: IEEE, pp. 363–368. isbn: 978-0-7695-3768-9. https://doi.org/10.1109/agile.2009.9

[19] Sypolt, Greg (2018). Avoiding Test Script Maintenance Nightmares. url: https://saucelabs.com/blog/avoiding-test-script-maintenance-nightmares (visited on 09/07/2019)

[20] Gundecha, Unmesh (2012). Selenium Testing Tools Cookbook. Packt Publishing

[21] Bucsics, Thomas et al. (2015). Basiswissen Testautomatisierung. Konzepte, Methoden und Techniken. second edition. dpunkt.verlag

[22] Micco, John (2016). Flaky Tests at Google and How We Mitigate Them. url: https://testing.googleblog.com/2016/05/flaky-tests-at-googleand-how-we.html (visited on 09/07/2019)

[23] Johner, Prof. Dr. Christian (2015). ISO 25010 und ISO 9126. url: https://www.johner-institut.de/blog/iec-62304-medizinische-software/iso9126-und-iso-25010/ (visited on 09/07/2019). https://doi.org/10.5220/0005216604050412

[24] Riley, Chris (2019). What's Next in Test Automation. url: https://saucelabs.com/blog/whats-next-in-test-automation (visited on 09/07/2019)

7 Authors

Lukas Krisper, is currently working as a Test Analyst at CAMPUSonline, Graz University of Technology. He leads the Community of Testing at the organisational unit, organises and advances the testing activities and processes. His current focus is on the automation of testing activities.

Markus Ebner, is currently working as a Researcher in the Department Educational Technology at Graz University of Technology. He deals with e-learning, mobile learning, technology enhanced learning and Open Educational Resources. His focus is on Learning Analytics at K-12 level. In addition, several publications in the area of Learning Analytics were published and workshops on the topic were held.

Martin Ebner, is with the Department Educational Technology at Graz University of Technology, Graz, Austria. (E-mail: martin.ebner@tugraz.at). As head of the Department, he is responsible for all university wide e-learning activities. He holds an Assoc. Prof. on media informatics and works at the Institute of Interactive Systems and Data Science as senior researcher. For publications as well as further research activities, please visit: http://martinebner.at . Email: martin.ebner@tugraz.at

Article submitted 2019-10-28. Resubmitted 2020-01-04. Final acceptance 2020-01-10. Final version published as submitted by the authors.